Appendices

Appendix A Join and meet algorithms

A.1 bottom-up join algorithm

By definition, the result of the join operation is a jrm. A jrm is defined by the intersection of its down set with the set of jims J. In our case, a jrm g is presented as the join of the members of a client specified subposet S. The jrm g is then defined by intersection of the down set of S with the set of jims J.

Since the construction "intersection of the down set of x with J" occurs frequently in the following, it is useful to define a notation for this set:

$J(x) = down(x) \cap J$

where x is either a member p of the lattice or a subset S of the lattice

In order to place g in the existing cover relation graph, we need to find the maximal lower bound and minimal upper bound for g in the existing graph. The maximal lower bound is the set of maximal members that are less than or equal to g while the minimal upper bound is the set of minimal members that are greater than or equal to g. If g is already in the graph, the maximal lower bound will contain a single member, the greatest lower bound, which must be equal to g.

Similarly, the minimal upper bound will contain a single member, the least upper bound, which also must be equal to g. If g is not in the graph, the maximal lower bound is the lower cover for g and the minimal upper bound is the upper cover for g.

Let S be a subset of the members of a lattice L. The bottom-up algorithm for computing g = join(S) is:

1. compute J(S)

2. compute the maximal lower bound of g from J(S)

3. compute the minimal upper bound of g from the minimal lower bound.

4. if g is not already in the graph, insert it between the minimal upper bound and the maximal lower bound

J(S) is computed by a straight forward traversal of the down set of S. A bottom-up algorithm for computing the maximal lower bound and an algorithm for computing the minimal upper bound follow.

A.1.1 compute the maximal lower bound:

The maximal lower bound is the set of maximal members of the down set of g.Thus to place g in the cover relation, we must compute the maximal members of the down set of g. This is the inverse of the usual definition of the down set, where we are given g and the cover relation and can find down-g by transitivity of the cover relation. Here, we are given only down-g ∩ J and must find all other members g' such that down-g' ∩ J is included in down-g ∩ J.

For notational convenience define J(g) = down-g ∩ J. Then in this notation, g' <= g iff J(g') is included in J(g).

The algorithm below depends on two lemmas:

Lemma 1:

J(g') is included in J(g) iff:

For g' a jrm, for all g" in the lower cover of g', J(g") is included in J(g).

For g' a jim, g' is a member of J(g)

Proof:

Part (a):

down-g' = {g'} ∪ down-g1" ∪...∪ down-gn"

J(g') = J({g'}) ∪ J(g1") ∪...∪ J(gn")

= J(g1") ∪...∪ J(gn") since J({g'}) = 0 if g' a jrm.

< J(g) iff each J(gi") < J(g)

Part (b):

This ia just a restatement of the definition of g, i.e. the Birkhoff representation theorem.

Lemma 2:

For every member g' of down-g, there is a chain j <: p1 <: p2 <: ... <: pn <: g', for all minimal members j of down-g, where "<:" is the cover relation.

Proof:

By definition j <= g' since g' is a member of down-g and j is a minimal member. The chain condition is just a statement of the fact that the order relation <= is the reflexive transitive closure of <:.

The client defines the jrm g by specifying a "join expansion", i.e. some subposet s. Then J(g) is defined to be the set of jims contained in a depth first traversal of the stored cover relation, starting at s. That is, J(g) is derived from the existing cover relation.

We maintain two subposets, gdown for the down set of g and gdown_max for the maximal members of the down set. Both are initialized to contain J(g). We then repeatedly "reduce" gdown_max, as follows:

Let gdown_max_up be the upper cover of gdown_max, i.e. the union of the upper covers of the members of gdown_max. Let g' be a member of gdown_max_up. If g' is a jrm and the lower cover of g' is included in gdown, then J(g') is included in J(g) (Lemma 1) and hence g' < g so we insert g' into gdown. If g' is a jim, g' < g iff it is already in gdown. In either case, g' is larger than any member of its lower cover and it should replace any members of its lower cover in gdown_max. So we remove any members of g'.lower_cover and insert g' in gdown_max. We continue until we can not reduce gdown_max any further, either because it has only a single member or because gdown_max_up contains no members with lower covers contained entirely in gdown.

At each step of the above process we have that gdown contains only members of the down set of g and at each step, the set of maximal members is reduced by replacing some members in it with larger members. The process does not complete until all the members of gdown_max are maximal. Further more, all members of down-g are reachable by this process (Lemma 2). So when the reduction completes, it contains the all maximal members of down-g that are present in the graph. If gdown_max contains a single member, it is g itself, otherwise gdown_max is the lower cover of g.

A.1.2 compute the minimal upper bound:

The algorithm for the minimal upper bound (mub) depends on the following lemma:

Lemma 3:

Define the minimal upper bound for any subposet S as:

mub(S) := min(intersection(up-s, for all s member of S))

Then join(S) <= mub(S)

Proof:

For each s, up-s is the set of all p such that s <= p. Then intersection(up-s) is the set of all p such that every s in S is <= p. But then join(S) <= p for all p (see Davey and Priestley Lemma 2.9.iii) and hence join(S) <= min(intersection(up-s)).

If join(S) < mub(S), then mub(S) is the upper cover of join(S), otherwise, obviously, join(S) = mub(S). Unfortunately, there is no simple condition that distinguishes the two cases. In particular, size of mub(S) = 1 is a necessary condition for equality, but it is not a sufficient condition. The necessary and sufficient condition is that the lower cover of join(S) = lower cover of mub(S).

We can compute mub(s) as follows:

Let s1, s2, ...sn denote the members of S. Define two cover sets C0 and C1. Initialize C0 to up-s0. Then:

```
for i= 2,n
  {
   c1 = up-si
   c0 = c0 intersection c1
  }
```

c0 = min(c0)

now c0 contains either the join or it's upper cover

A.2 top-down join algorithm

As above, let S be a subset of a lattice L, let g = join(S), and let M = mub(g).

The maximal lower bound (mlb) of g is contained in the lower cover of M. In fact, we have

mlb(g) = {p | p ∈ M.lower_cover and J(p) ⊆ J(g)}

Now the condition J(p) ⊆ J(g) will be satisfied if and only if the condition J(p') ⊆ J(g) for all p' ∈ p.lower_cover. In other words, the condition is recursive.

We can compute J(S) and mub(g) as above. Then we can compute mlb(g) top down as follows:

Initialize Q = M.lower_cover

For each member q of Q:

perform a depth first traversal of down(q). In a depth first traversal, there are three possible actions associated with each node in the graph: the previsit

action, the link action, and the postvisit action. Let p be the current node in the graph. Assume that each node p has a boolean variable is_contained that is true if $J(p) \subseteq J(g)$.

In the previsit action:

if p is a jim, set p.is_contained = J(g).contains_member(p)

else set p.is_contained = true

In the link action:

set p.is_contained = p.is_contained AND p'.is_contained

where p' is the node at the lesser end of the link.

When the traversal completes, if not q.is_contained, remove q from Q.

end for

Q now contains only those members m of M.lower_cover such $J(m) \subseteq J(g)$ and hence: mlb(g) = Q.

As above, if g is not already in the graph, insert it between the minimal upper bound and the maximal lower bound.

A.3 meet algorithm

Birkhoff representation theorem states that any jrm is equal to the join of the jims contained in it. Let $S = \{ l_0, l_1, .. l_n\}$ be a subset of a lattice L. Then the jims of $c = meet(S)$ are given by:

$$J(c) = \cap_{l \in s} J(s) = J(l_0) \cap J(l_1) \cap Y\, J(l_n)$$

so we have:

$$c = meet(S) = join(J(l_0) \cap J(l_1) \cap . . . J(l_n))$$

The meet operation thus reduces to a combination of set intersection operations and a join operation. The set intersection operation is standard, the join algorithms are given above.

Appendix B Cellular lattice constructor algorithm

A cellular lattice is constructed by making copies of a template and gluing the copies together. The copies are glued together by specifying which members in one copy are the same as members in another copy. The members which are thus glued together are the atoms of the template, the members of the template which form the upper cover of the bottom member.

The input to the cellular lattice constructor consists of:

template: any lattice

copy_ct: the number of copies to be made of the template

glue_ids: an array of arbitrary identifiers. The number of entries in this array must be equal to the number of atoms in the template times copy_ct

The algorithm requires 3 auxiliary data structures:

glue_map: a map from the externally defined glue_id to the id of a member in the result

template_map: a map from template member ids to result member ids.

lower_cover_stack: a stack containing lower cover sets

The top level of the algorithm is:

1. initialize glue_map to empty

2. create a top member for the result

3. for i = 0 to copy_ct:
   initialize template_map to empty.
   perform a depth first traversal of the template.

4. compute the minimal upper bound of the top of the result, as in A.1.2 above.

5. if the top of the result is not already in the graph, insert it below the

minimal upper bound.

The actions performed in the traversal are:

previsit action:

if the template member is not an atom, create an empty lower cover set for the copy of the template member and push it onto lower_cover_stack

link action:

get the result member corresponding to the template member at the lesser end of the link using template_map. This look up will always succeed because the depth first traversal ensures that the postvisit action of the lesser member will always occur before the member appears as the lesser member in a link.

insert the result member in the cover set that is on the top of lower_cover_stack

postvisit action:

if the current template member is a atom:

set glue_id equal to the next entry from glue ids and increment the glue_ids counter

look up glue_id in the glue_map. If an entry exists, glue_id has been used before, get the result id associated with it by the map. Otherwise, the

glue_id has not been used before, create a new result member and insert the (glue_id, result id) pair in the map.

insert the (current template id, result id) in template_map

if the current template member is not an atom:

Check to see if any members already in the block have a lower cover identical to what is currently on the top of lower_cover_stack. If there are any such members, they must be in the upper cover of every member of the cover on the stack, hence they must be in the upper cover of the first member. Check the lower cover of every member of the upper cover of the first member of the cover on the stack.

If a member with an identical lower cover already exists, it is the copy of the current template member. Otherwise, create a new result member and link it to the members of the top of lower_cover_stack

insert the (current template id, result id) in template_map

pop lower_cover_stack

if the current template member is the top of the template, insert result id into the lower cover of the top of the result